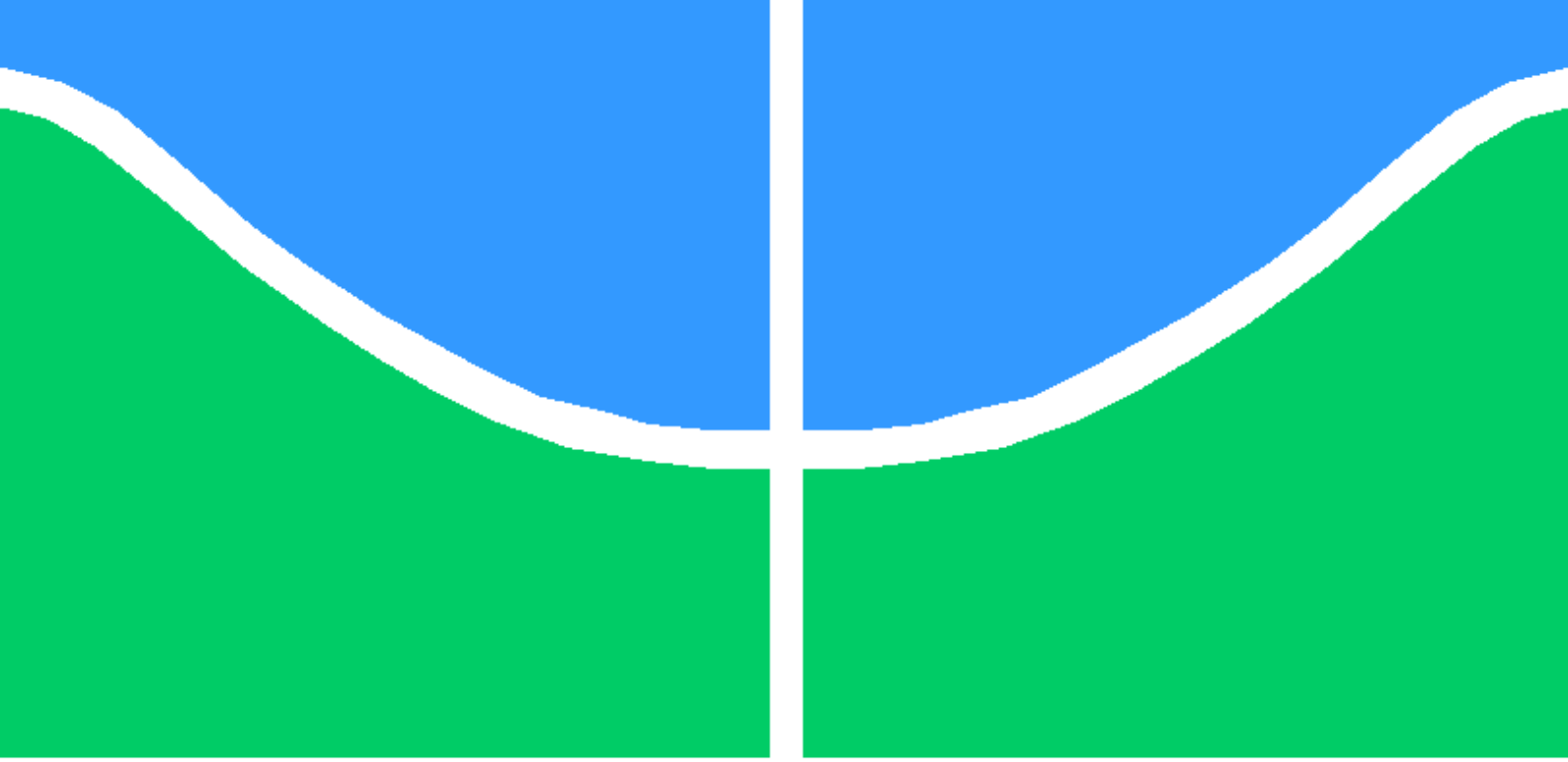

TRABALHO DE GRADUAÇÃO

APLICATIVO PARA MONITORAMENTO EM TEMPO REAL DA FROTA DE ÔNIBUS DE TRANSPORTE COLETIVO NO DISTRITO FEDERAL

Lucas Silva de Matos

Brasília, Julho de 2017

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

APLICATIVO PARA MONITORAMENTO EM TEMPO REAL DA FROTA DE ÔNIBUS DE TRANSPORTE COLETIVO NO DISTRITO FEDERAL

Lucas Silva de Matos

Relatório submetido ao Departamento de Engenharia Elétrica, como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Ugo Silva Dias, ENE/UnB _____
Orientador

Prof. Georges Daniel Amvame Nze, ENE/UnB _____
Examinador Interno

Prof. André Noll Barreto, ENE/UnB _____
Examinador Interno

Dedicatória

Dedico este trabalho a toda a minha família, pelo apoio e incentivo em todas as minhas decisões, e por todas as oportunidades que me proporcionaram para a minha formação acadêmica

Lucas Silva de Matos

Agradecimentos

Aos meus avós, que foram os principais resonsáveis pela minha criação. Obrigado por todos os valores ensinados e por nunca medirem esforços para me proporcionar as melhores condições de estudo possíveis, tornando possível esta formação acadêmica.

À minha mãe, que tenho certeza que está sempre ao meu lado intercedendo por mim.

Ao meu pai, meu grande companheiro, que nunca deixou de estar presente e me apoiou em todas as etapas da minha vida.

Ao meu tio, meu grande amigo e irmão, obrigado pela ajuda sempre solicita e pela sua companhia.

À minha irmã, que apesar da distância, foi fundamental no apoio às minhas decisões.

Ao Hudson de Moraes, pelo auxílio neste projeto, mesmo após o término de sua graduação.

Ao professor Ugo Dias, obrigado pelo auxílio no desenvolvimento deste projeto, sempre com dicas valiosas que foram fundamentais para a sua conclusão.

Aos meus grandes amigos que fiz nesta universidade, principalmente Paulo Marcelo, Yuri Freitas e Felipe Portela. Obrigado pela companhia durante estes anos acadêmicos.

Lucas Silva de Matos

RESUMO

Este trabalho pretende utilizar a tecnologia para melhorar o sistema de transporte público por ônibus, que carece de infraestrutura e oferece um serviço precário para a população. Foram criados três aplicativos, destinados tanto ao público geral, que utiliza este tipo de serviço, quanto aos gestores do sistema. O primeiro aplicativo seria utilizado pela população que utiliza o sistema de transporte público por ônibus, e disponibiliza ao usuário um sistema de consulta de rotas de ônibus em tempo real. O segundo aplicativo seria usado pelos responsáveis em cada veículo, e tem por objetivo enviar a um banco de dados externo periodicamente a localização atual do ônibus em questão, através de smartphones instalados no mesmo. O terceiro aplicativo seria utilizado pelos responsáveis pela gestão da frota de ônibus, com o objetivo de acompanhar e analisar o serviço, em tempo real. Estes aplicativos utilizam a tecnologia disponível nos smartphones para otimizar o sistema, identificando possíveis problemas e falhas, e para auxiliar a população na utilização deste serviço de transporte público.

Palavras-chave: Aplicativo, Android, Ônibus, Localização, Smartphone, Mobilidade, Transporte, Público

ABSTRACT

This study stands to use technology to improve the public transportation system, that lacks of infrastructure and offers a bad service for the population. Three mobile applications have been created, planned to help every person that needs to use the public bus service, and for the system managers. The first app should be used by the final user of the public transport system, and has as a main purpose the fetch of the best routes to travel to a final destination using this system. The second application should be used by the public transportation system workers, and has the purpose of sending to an external server and database the real-time bus location, with smartphones installed inside them. The third app should be used by the manager of the bus fleet, with the purpose of following and analyzing the system in real-time. These apps, together with the technologies present in today's smartphones, helps improving the system, identifying possible failures and problems, and helps every person in the usage of the public transportation system.

Key-words: Application, Android, Bus, Location, Smartphone, Mobility, Transport, Public

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Acrônimos

*A*OSP	Android Open Source Project
*A*PI	Application Program Interface
*B*aaS	Backend as a Service
*B*I	Business Intelligence
*B*SON	Binary JSON
*G*PS	Global Positioning System
*H*TTP	Hypertext Transfer Protocol
*H*TTPS	Hyper Text Transfer Protocol Secure
*I*D	Identity
*I*DE	Integrated Development Environment
*J*SON	JavaScript Object Notation
*N*oSQL	Non SQL
*P*aaS	Platform as a Service
*P*HP	Hypertext Preprocessor
*S*DK	Software Development Kit
*U*RL	Uniform Resource Locator
*X*ML	Extensible Markup Language

Capítulo 1

Introdução

1.1 Objetivos do Projeto

O Sistema de Transporte Público em boa parte do Brasil é precário e atrasado, devido principalmente a problemas de gestão, investimento e políticas públicas do setor. Usuários que utilizam deste meio de locomoção passam por dificuldades todos os dias, como ônibus lotados e tempo exagerado de espera nos pontos de ônibus. O planejamento e organização tornam-se então extremamente necessários para que a população usufrua de um sistema de transporte público que atenda as suas necessidades.

1.2 Visão Geral

Para que o objetivo descrito fosse atingido, foram criados os três aplicativos seguintes, com funcionalidades e utilizações diferentes:

- O primeiro deles será utilizado pelo usuário final, e deve ser simples, deve possuir uma interface agradável e deve ter um bom desempenho, para que o usuário tenha uma boa experiência. Nele será apresentado um mapa, com sua localização e os pontos de ônibus mais próximos. Em cada ponto de ônibus será apresentado uma lista com o tempo de chegada dos ônibus que chegarão neste ponto, listados em ordem de tempo de chegada. Será possível selecionar um ônibus para acompanhar sua trajetória, e também gerar um alerta vibratório quando este estiver a um minuto da chegada. Além disto, o usuário pode selecionar uma rota para o seu destino e visualizar quais ônibus fazem este trajeto.

- O segundo deve ser utilizado pelos funcionários deste sistema, com a finalidade de prover as informações atualizadas acerca deste. Para o cálculo correto da estimativa de chegada do ônibus é necessária a informação da localização dos ônibus, de forma sempre atualizada e real. Aliando esta informação e com o auxílio da Google Maps API, o cálculo da estimativa

de tempo pode ser feito. Como o aplicativo não possui as informações reais acerca das localizações dos ônibus do sistema público de transporte, e para que as simulações dos aplicativos possam ser feitas, foram simuladas rotas realizadas por ônibus de linhas aleatoriamente escolhidas. Este aplicativo foi desenvolvido pelos alunos Lucas Matos e Hudson de Moraes, e será apresentado neste trabalho.

- O terceiro deve ser utilizado pelos responsáveis da gestão da frota de ônibus do Distrito Federal, para monitorar as rotas e os ônibus, afim de melhorar os serviços prestados. Com este aplicativo é possível consultar informações como placa, modelo, número de infrações, sobre cada veículo da frota, e também acompanhar em tempo real a localização deles, assim como sua estimativa de chegada às paradas de ônibus. Este aplicativo foi desenvolvido pelo aluno Lucas Matos, e será apresentado neste trabalho.

1.3 Aplicações de Gestão de Frotas semelhantes

São apresentadas a seguir algumas aplicações semelhantes às desenvolvidas neste projeto. Este projeto visa aplicar algumas destas características de gestão de frota, com algumas modificações para se adequar ao sistema de trasporte público por ônibus, e adiciona outras funcionalidades importantes, que serão apresentadas neste trabalho. Todos os produtos encontrados estão disponíveis no mercado para pessoas físicas. Produtos destinados especificamente à empresas que prestam o serviço de transporte público por ônibus não foram encontrados, talvez devido a restrição dos mesmos.

1.3.1 Vivo Gestão de Frotas

O Vivo Gestão de Frotas foi criado para atender empresas privadas que possuam uma pequena frota de veículos e que desejam reduzir custos, que segundo a fabricante, pode ser de até 30%. Para utilizar o produto, rastreadores são instalados em cada um dos veículos, ao custo de R$ 100,00. O custo mensal com o produto não é informado no site da fabricante. Este serviço foi criado em 2014, sendo este o primeiro produto neste setor por uma operadora de telefonia.

Esta solução é um sistema desenvolvido para fazer o rastreamento de veículos. A Figura 1.1 mostra a tela inicial do aplicativo, onde é possível selecionar o veículo a ser localizado. Além do serviço de rastreamento, este produto possui diversas outras funcionalidades:

- Determinar uma região em que o veículo possa transitar.
- Estipular um limite de velocidade para o veículo.
- Gerar relatórios com o histórico de trajetos percorridos por cada veículo.
- Minimizar despesas operacionais.
- Promover manutenção preventiva dos veículos.

Após selecionar o veículo, é apresentado em um mapa a localização do mesmo, como mostrado na Figura 1.2.

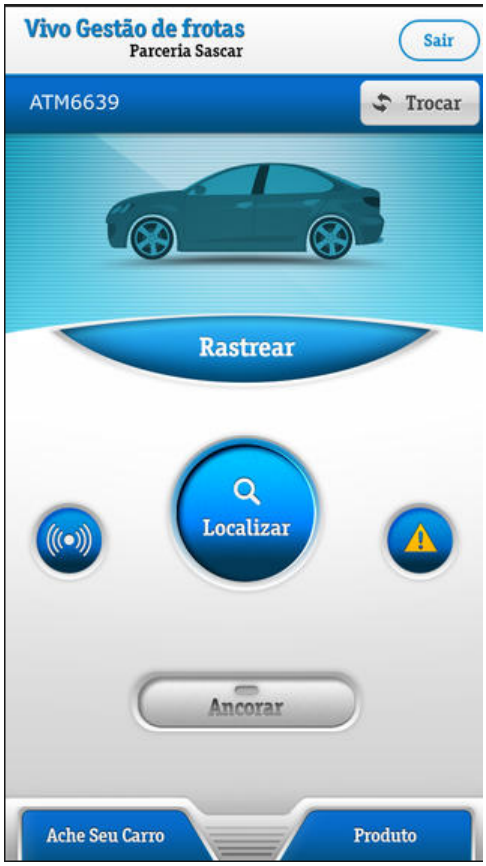

Figura 1.1: Tela inicial App Vivo

Figura 1.2: Tela de rastreio App Vivo

1.3.2 Frota+

A montadora Iveco Bus lançou em 2016 o Iveco Frota+, um aplicativo destinado à gestão da frota de ônibus de empresas. Apesar de ser destinado às companhias que prestam um serviço de transporte privado, ele pode ser utilizado para o controle de frotas de qualquer segmento. O aplicativo está disponível para o sistema operacional Android, com *download* na loja Google Play.

O aplicativo possui vantagens para se controlar os custos operacionais da frota de uma empresa, como por exemplo o monitoramento do consumo de combustível e a eficiência por quilômetro rodado dos veículos. É possível também programar o alerta de manutenção preventiva e gerar relatórios com informações e gráficos sobre o faturamento e finanças.A Figura 1.3 apresenta a tela inicial, onde é possível acessar as funcionalidades do aplicativo.

A Figura 1.4 mostra a frota de veículos cadastrados no aplicativo, com suas informações de placa, modelo e ano.

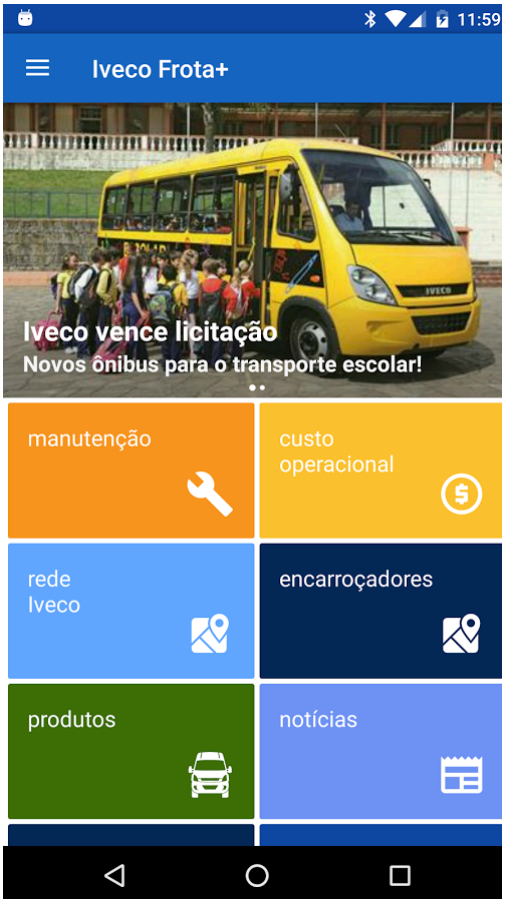

Figura 1.3: Tela inicial Frota+

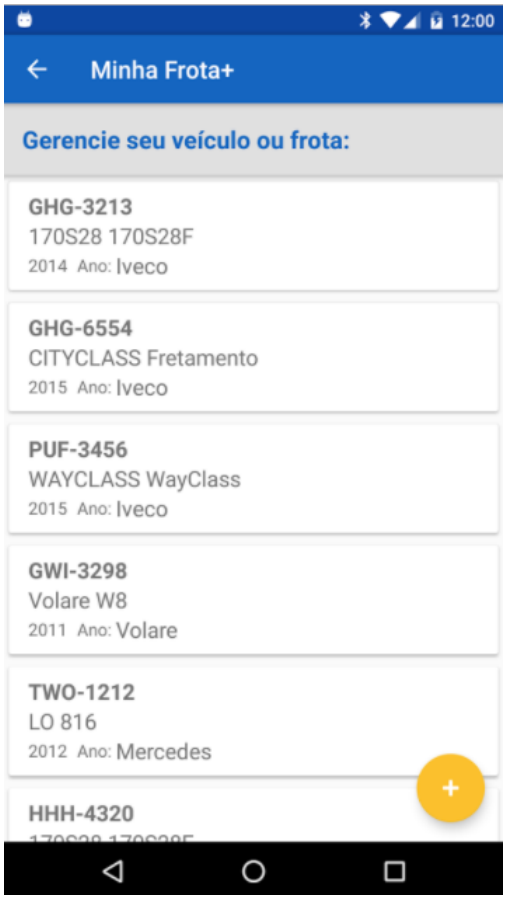

Figura 1.4: Tela veículos Frota+

Ao selecionar a opção sobre o Custo Operacional, mostrado na Figura 1.5, são apresentadas informações e gráficos sobre a Quilometragem Média Rodada por cada veículo.

Quando selecionada a opção Manutenção, o aplicativo apresenta quatro campos:

- Minha Agenda de Manutenção: Mostra a previsão das manutenções de cada veículo, em um calendário.
- Óleos e fluidos recomendados: Apresenta uma lista de produtos recomendados para cada tipo de veículo.
- Plano de Manutenção Iveco: Lista um modelo de manutenção utilizado pela empresa Iveco.
- Perguntas Frequentes: Conjunto de perguntas e respostas sobre manutenção de veículos.

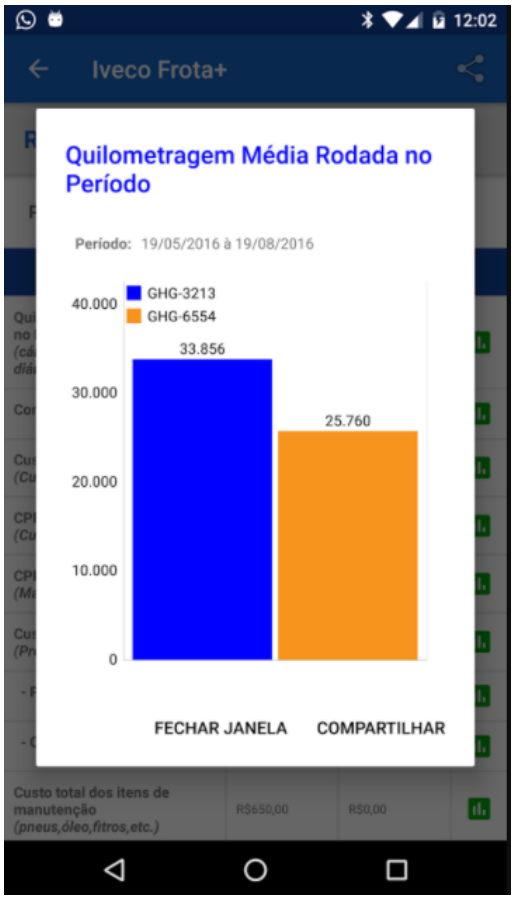

Figura 1.5: Tela Custo Operacional Frota +

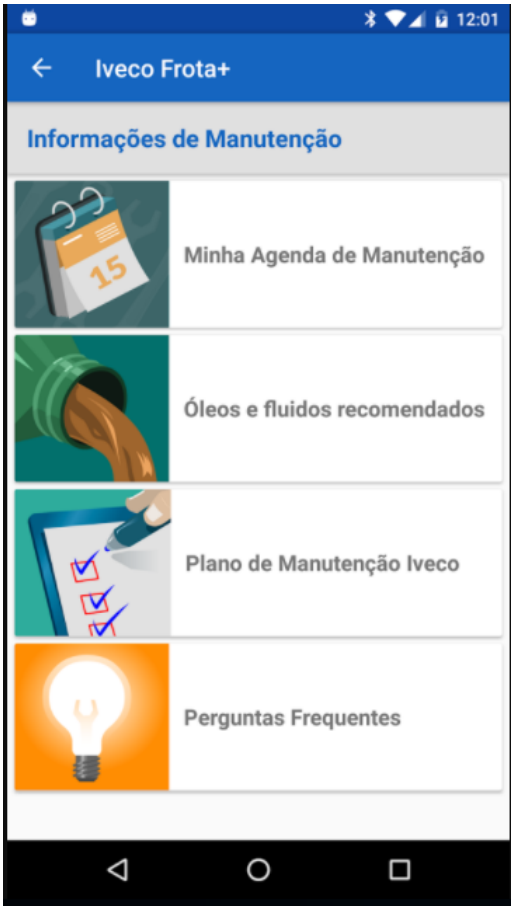

Figura 1.6: Tela Manutenção Frota+

1.3.3 Oi Gestão de Frotas

É a solução da empresa Oi para monitoramento e rastreamento de veículos via *G*PS e dados móveis. Permite a gestão e otimização dos recursos pelo aumento da produtividade, suprindo a necessidade de empresas privadas que possuem frotas de veículos. Para utilizar o produto, rastreadores são instalados em cada um dos veículos, ao custo de R$ 70,00. O custo mensal com o produto não é informado no site da fabricante. Seguem as principais características desta solução:

- Gestão online de frotas
- Controle de revisão e manutenção preventiva
- Acompanhamento do perfil do condutor através de informações de km e rotas
- Controle de quilômetros percorridos
- Controle de consumo e gasto de combustível
- Controle de infrações cometidas pelos motoristas
- Sistema de mensagem instantânea para comunicação com o motorista
- Controle da região trafegada

No mapa do aplicativo é possível visualizar a localização de todos os veículos da frota, como mostrado na Figura 1.7. No mapa, estes veículos são identificados por diferentes cores:

- Azul: Ignição ligada
- Vermelha: Ignição desligada
- Cinza: Veículo sem transmissão acima de 5 dias

Ao se cliclar no ícone centralizado do mapa, é possível visualizar as informações sobre as infrações cometidas pelos veículos da frota. Ao se visualizar as infrações de cada veículo pode-se realizar contato com os condutores por meio de ligação ou mensagem, como mostrado na Figura 1.8.

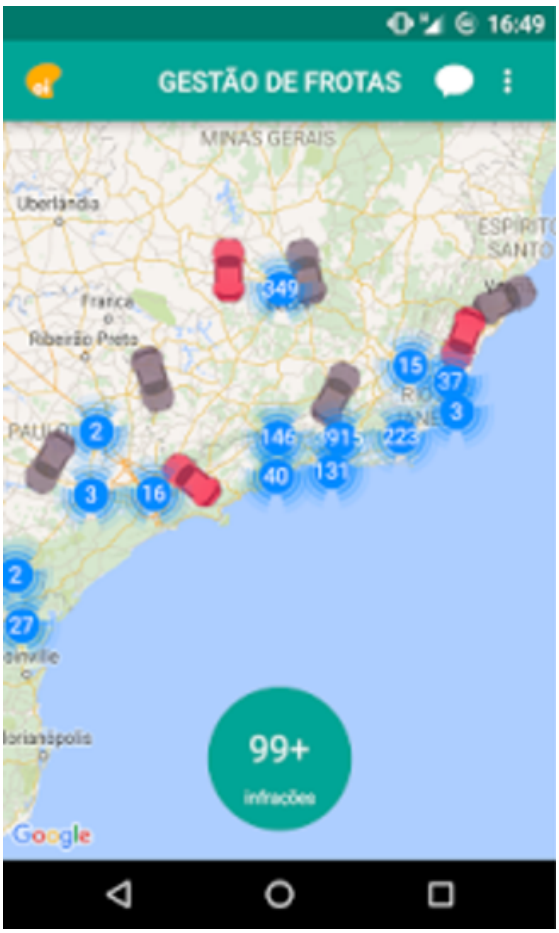

Figura 1.7: Tela Mapa App Oi

Figura 1.8: Tela Infrações App Oi

Na tela inicial do aplicativo, mostrada na Figura 1.9, encontramos algumas funcionalidades de utilização do condutor:

- Check-in: Permite que o condutor registre o início/fim de sua jornada de trabalho, atividades e horário de almoço.
- Mensagem: Permite que o condutor se comunique com o Gestor da Frota, quando for solicitado
- Perfil: Permite visualizar as infrações cometidas pelo condutor e sua respectiva pontuação. Além de representar numericamente as infrações cometidas pelo condutor, esta pontuação também é utilizada para determinar a posição do condutor no Ranking.
- Ranking: Apresenta a posição dos condutores entre os demais condutores da frota. O *Ranking* é ordenado do maior para o menor infrator

Figura 1.9: Tela Inicial Oi Gestão de Frotas

1.4 Diferenciais

Todos os produtos apresentados neste trabalho são destinados à empresas que desejam realizar a gestão de sua frota com veículos de qualquer porte. Este trabalho, no entanto, visa apresentar uma solução personalizada para o sistema de transporte público por ônibus. Apesar de possuir algumas funcionalidades semelhantes, toda a parte de monitoramento da frota, com o mapeamento das paradas de ônibus, e as estimativas de tempo de chegada, são pontos inovadores neste projeto.

1.5 Metodologia

As etapas da metodologia aplicada ao projeto são descritas a seguir:

A Etapa 1 consiste na realização de um estudo sobre o desenvolvimento de aplicações para Android, assim como um estudo da linguagem de programação Java.

A Etapa 2 compreende a pesquisa das melhores ferramentas a serem utilizadas para o desenvolvimento do aplicativo.

A Etapa 3 trata da formulação escrita do projeto final de graduação 1, que contém o planejamento realizado a partir da primeira etapa de pesquisa, em que é exposto uma descrição das

atividades a serem realizadas e o seu cronograma de execução.

Na Etapa 4 é realizado o desenvolvimento do aplicativo.

A Etapa 5 trata da formulação escrita e apresentação do projeto final de graduação 2, contendo os resultados obtidos e a versão final do aplicativo.

Capítulo 2

Fundamentação Teórica

2.1 Android

O Android é um sistema operacional baseado no Linux, desenvolvido pelo AOSP e gerenciado pelo Google. Constantemente atualizado, sua última versão lançada é a 7.0, chamada Nougat. Devido a limitações das funcionalidades e do *smartphone* utilizado para testes no desenvolvimento desse projeto, a versão utilizada foi a 5.0, Lollipop. O sistema operacional Android é dividido nas seguintes quatro áreas:

- Applications: São os aplicativos que temos contato no uso de um smartphone.
- Framework: É a API que permite a interação das aplicações com o sistema Android.
- Libraries: Bibliotecas que fornecem um suporte para as funções do *Framework*, como banco de dados.
- Linux Kernel: Responsável pela comunicação do sistema com o *hardware* do *smartphone*.

Uma aplicação Android, como as desenvolvidas nesse projeto, é um simples programa instalável que pode ser executado em um smartphone com uma versão atualizada do Sistema Android. Essa aplicação consiste nos cinco componentes a seguir:

- Application: É a base da aplicação e contém os códigos java criados pelo desenvolvedor.
- Activity: É a representação visual do aplicativo. Podem ser criadas várias dessas atividades em cada aplicativo, e possui como base o XML.
- Service: É um componente independente do código da aplicação e das atividades, que pode executar diversas tarefas de interação com o usuário, como a função de *Vibracall* do *smartphone*.

- Broadcast receiver: É utilizado para registrar certos eventos ou mensagens do sistema ou da aplicação, como o recebimento de uma ligação ou mensagem de texto.
- Content provider: Fornecem a interface e gerenciam o acesso aos dados da aplicação.

2.2 Plataforma

O desenvolvimento de aplicativos para Android pode ser feito através de uma IDE, chamada Android Studio, desenvolvida pelo Google, para a criação de aplicações utilizando a linguagem de programação orientada a objetos Java. A escolha desta plataforma foi feita devido à facilidade de se encontrar material de apoio para o desenvolvimento do aplicativo, além de estar disponível para os principais sistemas operacionais. O sistema Android está presente na maior parte dos *smartphones* de hoje e por isso o alcance e sucesso do aplicativo pode ser maior.

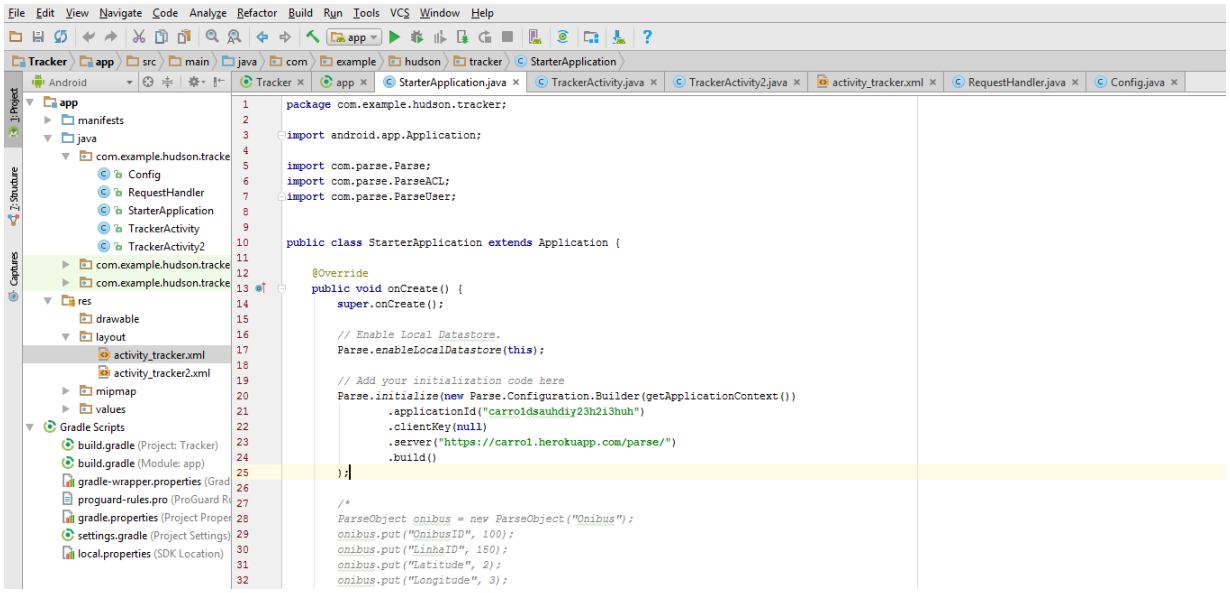

Figura 2.1: Android Studio

2.3 Aplicativos criados

Fazem parte do escopo deste trabalho dois aplicativos:

1) Transmissor: Esta proposta de aplicativo deve funcionar em smatphones que seriam instalados em todos os ônibus da frota. Ele irá funcionar como um transmissor, que através da rede de dados móvel, enviará a cada 3 segundos a localização e velocidade atual do veículo para o banco de dados.

2) Gestão da frota: Aplicativo que será usado pelo responsável, dentro da empresa licitante do sistema de transporte público, para monitorar em tempo real os veículos da frota. Ele possuirá uma conexão com o banco de dados onde será possível acessar as informações enviadas pelo aplicativo transmissor.

2.4 Google Maps Directions API

A Google Maps Directions API é uma das principais APIs usadas neste projeto. Nela encontramos todas as funcionalidades presentes nos produtos *Maps* do Google, onde é possível traçar rotas entre dois pontos, calcular tempo estimado de chegada, através de diferentes meios de locomoção, e informações sobre trânsito.

Ela permite o cálculos de rotas para diferentes meios de transporte: transporte público, condução, caminhada ou bicicleta. Nestes trajetos, há duas possibilidades diferentes de especificar origens, destinos, podendo ser feitas se utilizando do nome dos locais desejados, ou de suas posições geográficas (latitude e longitude).

Esta requisição é feita através de uma solicitação HTTP, através de uma URL no seguinte formato:

http://maps.googleapis.com/maps/api/directions/output?parameters

Para que se tenha uma maior segurança na solicitação desses dados, foi utilizado o HTTPS, que proporciona maiores garantias de segurança a todas as requisições. A URL fica na seguinte forma:

https://maps.googleapis.com/maps/api/directions/output?parameters

Os valores de saída e os parâmetros devem ser escolhidos de acordo com as necessidades no cálculo da rota.

Para os valores de saída temos duas opções:

- json – as informações retornadas serão no formato JSON.
- xml – as informações retornadas serão no formato XML.

Para este projeto foi escolido o formato JSON. A Figura 2.2 mostra um trecho da resposta de uma requisição HTTP entre os endereços "SHIGS 703"e "SHIGS 705"à Google Maps API.

https://maps.googleapis.com/maps/api/directions/json?origin=SHIGS%20703&destination=SHIGS%20705&key=AIzaSyCoA45dCPi3AoNvtFR52A_sd2D-ezjJVvM

```
"routes" : [
   {
      "bounds" : {
         "northeast" : {
            "lat" : -15.8004282,
            "lng" : -47.8947306
         },
         "southwest" : {
            "lat" : -15.8063058,
            "lng" : -47.9007995
         }
      },
      "copyrights" : "Dados cartográficos ©2017 Google",
      "legs" : [
         {
            "distance" : {
               "text" : "1,6 km",
               "value" : 1580
            },
            "duration" : {
               "text" : "5 minutos",
               "value" : 288
            },
            "end_address" : "St. de Habitações Individuais Geminadas Sul 705 - Asa Sul, Brasília - DF, Brasil
            "end_location" : {
               "lat" : -15.8063058,
               "lng" : -47.90027629999999
            },
            "start_address" : "St. de Habitações Individuais Geminadas Sul 703 - Asa Sul, Brasília - DF, Bras
            "start_location" : {
               "lat" : -15.8012314,
               "lng" : -47.8966007
            },
            "steps" : [
               {
                  "distance" : {
                     "text" : "0,3 km",
                     "value" : 298
                  },
                  "duration" : {
                     "text" : "2 minutos",
                     "value" : 102
                  },
                  "end_location" : {
                     "lat" : -15.8015639,
                     "lng" : -47.8947306
                  },
                  "html_instructions" : "Siga na direção \u003cb\u003enordeste\u003c/b\u003e",
                  "polyline" : {
                     "points" : "tdm_BvxicHUS[U[WqAiAn@w@n@{@h@s@V[JMHIBCDABADAFALA"
                  },
                  "start_location" : {
                     "lat" : -15.8012314,
                     "lng" : -47.8966007
                  },
                  "travel_mode" : "DRIVING"
               },
```


Figura 2.2: Resposta do Google Maps API

2.5 Parse Server

Criada em 2011 pelos americanos Tikhon Bernstam, Ilya Sukhar, James Yu e Kevin Lacker, a empresa Parse disponibilizava SDK's para o desenvolvimento de *backends* móveis para Windows 8, Windows Phone 8, iOS, Android, Javascript e OS X. Foi considerada uma das 50 empresas mais inovadores do mundo, no ano de 2013. A visibilidade e o reconhecimento adquiridos pela empresa chamaram a atenção do Facebook, que neste mesmo ano fez uma oferta de $85 milhões, adquirindo assim a companhia.

O sistema Parse é considerado um BaaS, isso é, um modelo que disponibiliza aos desenvolvedores uma forma de conectar seus aplicativos à um banco de dados na nuvem, deixando transparente toda a estrutura necessária para essa comunicação. O sistema Parse original foi desativado pelo Facebook no dia 29 de janeiro de 2017. Por isso, foi disponibilizado para seus usuários uma versão *open source*, que foi utilizado nesse projeto de graduação. Esta versão possui todas as funcionalidades de comunicação do Parse original, mas deve ser utilizado em conjunto com um sistema em nuvem com um banco de dados que possa rodar Node.js. A Figura 2.3 resume o sistema Parse completo, antes da desativação realizada pelo Facebook, e a Figura 2.4 apresenta a estrutura do Parse *open source* com o sistema utilizado em nuvem, e o banco de dados.

Figura 2.3: Estrutura antiga Parse

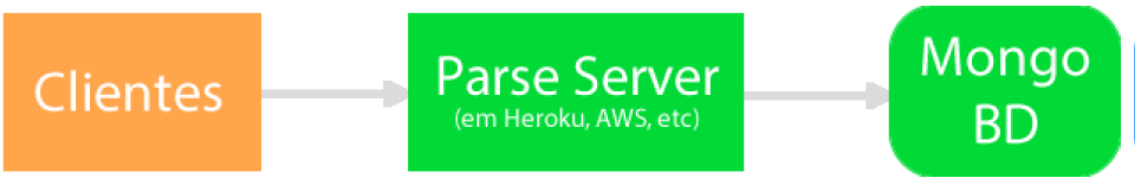

Figura 2.4: Estrutura nova Parse

A maioria das aplicações armazena dados e interage com serviços hospedados na internet. Essas aplicações necessitam de requisitos complementares para realizar esta comunicação. Por muito tempo desenvolvedores precisavam criar essas estruturas individualmente, sendo necessário conhecimento de diversas áreas, como redes, *hardware*, servidores e desenvolvimento. É com o intuito de simplificar esse processo que foi criado o Parse.

O Parse é uma ferramenta que disponibiliza ao usuário um sistema simplificado de comunicação entre um aplicativo móvel e um sistema na nuvem. Nele, todo o processo de comunicação e troca de

mensagens entre o aplicativo e o banco de dados é transparente, como a troca de mensagens JSON, com as informações trocadas, e a infraestrutura de armazenamento dos dados e de apresentação para o usuário.

2.6 Heroku/MongoDB

O sistema Heroku é um PaaS, que se enquadra na categoria de serviços em nuvem, que fornece aos usuários toda a infraestrutura necessária para o desenvolvimento, teste e gerenciamento de aplicações. Com o suporte para aplicações escritas em Ruby, Node.js, Java, Python, Clojure, Scala, Go e PHP, o Heroku é uma ferramenta muito poderosa para se trabalhar com aplicações na nuvem.

Em conjunto será utilizado um sistema open source chamado MongoDB, um banco de dados que substitui as funcionalidades referentes ao armazenamento que não estão disponíveis na versão open source do Parse utilizada. O MongoDB armazena dados em um formato chamado BSON (Binary JSON), e é classificado como um programa de banco de dados NoSQL, pois ele evita a estrutura tradicional de banco de dados baseados em tabelas. A figura 2.5 apresenta a estrutura com as três plataformas utilizadas, Parse, Heroku e MongoDB.

Figura 2.5: Estrutura utilizada

2.7 Banco de Dados

Como descrito nos tópicos anteriores, foram utilizados em conjunto o sistema Parse *open source*, com o Heroku e o MongoDB. Para gerenciar todo o sistema de banco de dados foi utilizado o Parse Dashboard. Para utilizá-lo, foi necessário instalar o pacote que contêm o Dashboard e o Node.js, que é uma plataforma baseada em JavaScript e utilizada para construir aplicações. Para utilizar a *dashboard*, deve-se iniciar o *prompt* de comando do sistema operacional e digitar o seguinte comando:

parse-dashboard –appId carro1dsauhdiy23h2i3huh –masterKey 2i1h3i12478yfhs7y8da –serverURL "https://carro1.herokuapp.com/parse--appName Carro

Para abrir a Dashboard é necessário digitar no campo *URL* de qualquer navegador o seguinte endereço:

http://localhost:4040

A figura 2.6 apresenta a *dashboard* em funcionamento.

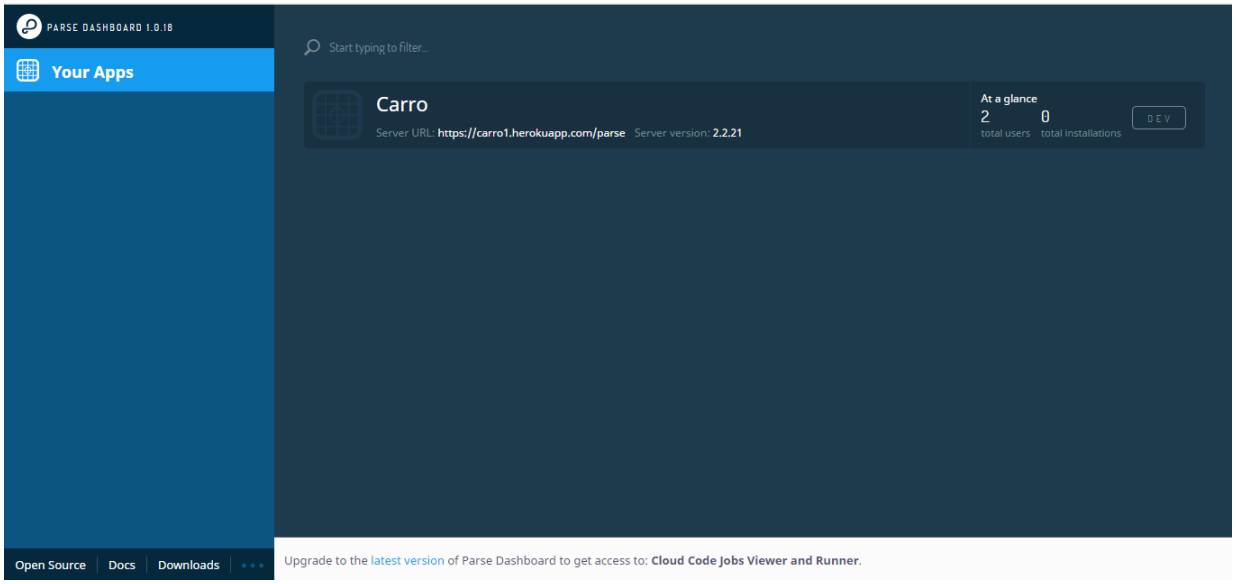

Figura 2.6: Tela inicial Parse Dashboard

A Figura 2.7 exibe o banco de dados utilizado no projeto, nomeado de Ônibus. Foi selecionado somente uma linha de ônibus como exemplo, a linha 102, com um só ônibus. São apresentados em conjunto informações sobre o veículo exibidas no aplicativo do Gestor.

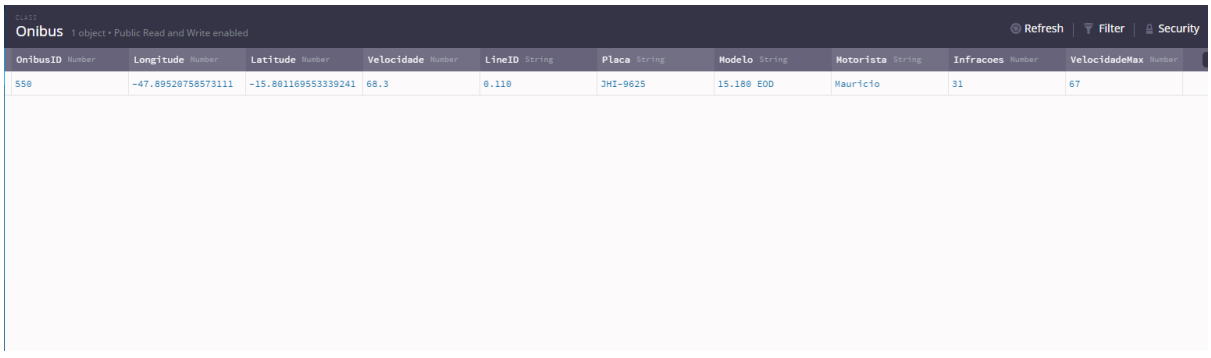

Figura 2.7: Banco de Dados Parse Dashboard

2.8 Estrutura dos Apps

A seguir serão detalhadas as estruturas dos dois aplicativos criados neste trabalho, o Gestão de Frotas e o Transmissor. Como detalhado anteriormente, foi utilizada a plataforma Android Studio

IDE, que é baseada na linguagem de programação java. Foram incluídos alguns trechos do código java e xml, que estão inclusos como imagens neste trabalho para um melhor entendimento dos procedimentos realizados. Somente alguns trechos do código criado foram incluídos neste capítulo, para que o texto não se torne muito extenso e para que os exemplos sejam claros para o leitor.

Para se conectar ao banco de dados e utilizar a estrutura do Parse, a primeira classe executada pelos dois aplicativos é a *StarterApplication*. São importados três pacotes para se utilizar as funções do Parse.

```
import com.parse.Parse;
import com.parse.ParseACL;
import com.parse.ParseUser;
```

Figura 2.8: Código Pacotes Parse

É necessária a especificação dos parâmetros do projeto criado no Heroku, para a conexão do aplicativo com o banco de dados. Para isso foram detalhadas as informações a seguir:

```
// Enable Local Datastore.
Parse.enableLocalDatastore(this);

// Add your initialization code here
Parse.initialize(new Parse.Configuration.Builder(getApplicationContext())
        .applicationId("carro1dsauhdiy23h2i3huh")
        .clientKey(null)
        .server("https://carro1.herokuapp.com/parse/")
        .build()
```

Figura 2.9: Código Credenciais Heroku

- applicationID("carro1dsauhdiy23h2i3huh"): Indica o ID único da aplicação no Heroku.
- server("https://carro1.herokuapp.com/parse/"): Indica a URL da aplicação no servidor.

Após a especificação dos parâmetros, são executados alguns métodos padrão para o correto estabelecimento da conexão, apresentado na Figura 2.10.

```
ParseUser.enableAutomaticUser();
ParseACL defaultACL = new ParseACL();
// Optionally enable public read access.
// defaultACL.setPublicReadAccess(true);
ParseACL.setDefaultACL(defaultACL, true);
```

Figura 2.10: Código Conexão Parse

2.8.1 Tracker

Após a conexão com a nuvem descrita anteriormente, a classe *TrackerActivity* é iniciada. Nela foram criados campos onde serão digitadas a linha do ônibus e sua id. Para isso foram adicionados os campos *EditText* no arquivo XML. A Figura 2.11 mostra o XML de um dos campos descritos.

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="text"
    android:ems="10"
    android:id="@+id/editlinhaonibus"
    android:layout_gravity="center_horizontal" />
```

Figura 2.11: XML Campo Linha Ônibus

Após esse preenchimento, o usuário deve clicar no botão "INICIAR", para que o aplicativo comece a enviar as informações sobre a localização e velocidade do veículo para o banco de dados. Para isso foi criado um "Button", no arquivo xml, e um *setOnClickListener*, no código java, que tem a função de acionar a classe *TrackerActivity2*, quando o botão é clicado. Os dois trechos dos códigos são descritos nas Figuras 2.12 e 2.13.

```
<Button
    android:id="@+id/btnStartLocation"
    android:background="@color/btnStart"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="60dp"
    android:text="Iniciar"
    android:textSize="25sp"
    android:textColor="@color/btnFont"
    android:paddingLeft="20dp"
    android:paddingRight="20dp"
    android:layout_gravity="center_horizontal" />
```

Figura 2.12: XML Button

```
btnStartLocation.setOnClickListener((v) → {

        String linha = editlinhaonibus.getText().toString();
        String id = editidonibus.getText().toString();

        Intent myIntent = new Intent(TrackerActivity.this, TrackerActivity2.class);
        myIntent.putExtra("linha",linha);
        myIntent.putExtra("id",id);
        TrackerActivity.this.startActivity(myIntent);

});
```

Figura 2.13: Código Java Button

Com o clique do botão "INICIAR", a classe *TrackerActivity2* é iniciada. Esta possui a função de monitorar a localização e a velocidade do ônibus periodicamente. O método *getMyLocation* é responsável pela atualização do banco de dados em tempo real. A Figura 2.14 mostra o trecho do código que envia para o banco de dados as informações de Latitude, Longitude e a Velocidade do ônibus. Para o correto funcionamento do aplicativo, é necessário que o método *getMyLocation* seja acionado assim que a classe *TrackerActivity2* for iniciada, isso é, dentro do método *onCreate* (responsável pela primeira atualização do banco de dados), e também em outro método criado, o *onLocationChanged*. Este método é responsável pela atualização periódica do banco de dados, sendo acionado todas as vezes que a localização do usuário mudar.

```
for (ParseObject Onibus : objects) {
    Onibus.put("Latitude", myLocation.getLatitude());
    Onibus.put("Longitude", myLocation.getLongitude());

    if (myLocation.getSpeed() == 0) {

        Onibus.put("Velocidade", 0.1);

    } else {
        Onibus.put("Velocidade", (myLocation.getSpeed()*3600/1000));
    }
```

Figura 2.14: Atualização Dados no Banco de Dados

2.8.2 Gestor

Após a conexão com a nuvem, assim como o aplicativo *Tracker*, a classe *MainActivity* é iniciada. Nela está presente a tela inicial do *app*, onde temos o sistema de *Login*. Foram adicionados dois campos, usuário e senha, para ser efetuada a autenticação do gestor. Ao digitar as credenciais únicas e clicar no botão, é acionado o método que verifica junto ao banco de dados se as informações estão corretas. Caso o resultado for falso, é gerada uma mensagem de erro na tela, e caso positivo, o aplicativo é iniciado, junto à classe *BusLocation*.

```
if (validarUsuario == true && validarSenha == true) {
    redirectUser();
} else {
    if (bUsuario == null || bSenha == null || validarSenha == false || validarUsuario == false){

        AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
        builder.setMessage("Login Falhou")
                .setNegativeButton("Tente Novamente", null)
                .create()
                .show();
    }
}
```

Figura 2.15: Método que valida o Login

A classe *BusLocation* é criada utilizando uma *Activity* do Google Maps inclusa no Android Studio IDE, onde é inicializada uma estrutura completa de Mapa criado pelo Google, que permite utilizar todas as funcionalidades da Google Maps API. Todos os pacotes e funções necessários são importados automaticamente. Todo o mapa é iniciado dentro do método *onCreate*, que é o primeiro a ser iniciado em todas as classes.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_bus_location);
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
```

Figura 2.16: Método onCreate da classe BusLocation

Junto ao mapa, é iniciado um campo de pesquisa *EditText* e um *Button*. Ambos possuem a função de fazer a pesquisa da linha de ônibus que se deseja monitorar. A Figura 2.17 mostra a inclusão do *Button* no arquivo XML.

```
<Button
    android:id="@+id/search_button"
    android:layout_width="90dp"
    android:layout_height="50dp"
    android:layout_weight="0.5"
    android:onClick="onMapSearch"
    android:text="Search"
    android:layout_alignBaseline="@+id/editText"
    android:layout_alignBottom="@+id/editText"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```

Figura 2.17: XML Button

A linha de ônibus é digitada nesse campo de pesquisa e é comparada no banco de dados, que retorna o campo *location*, com a informação da localização do(s) ônibus desta linha. A figura 2.18 mostra o código que utiliza das funções do Parse para verificar essas informações no banco de dados.

Após retornadas essas informações, é iniciado o processo de verificação junto à Google Maps API de diversas informações sobre a linha desejada. O primeiro método acionado é o *getUrl*, onde são enviados como parâmetros as informações retornadas pelo banco de dados, a localização do ônibus e o destino final da linha. Como explicado anteriormente, é necessário criar uma URL, que irá fazer a requisição HTTP. A função desse método é criar uma URL específica para a linha desejada, que deve ter o seguinte formato:

https://maps.googleapis.com/maps/api/place/autocomplete/output?parameters

```
ParseQuery<ParseUser> userQuery = ParseUser.getQuery();
userQuery.whereEqualTo("linha", linha);
userQuery.findInBackground((objects, e) → {
        if (e == null) {

            if (objects.size() > 0) {

                for (ParseUser driver : objects) {
                    driverLocation = driver.getParseGeoPoint("location");
                }

            }
        }
});
```

Figura 2.18: Código que retorna a localização do Banco de Dados

A Figura 2.19 mostra os parâmetros presentes na requisição HTTP.

```
private String getUrl(LatLng origin, LatLng dest) {

    String str_origin = "origin=" + origin.latitude + "," + origin.longitude;
    String str_dest = "destination=" + dest.latitude + "," + dest.longitude;
    String str_mode = "mode=transit";
    String str_transit_mode = "transit_mode=bus";
    String str_key = "key=AIzaSyCoA45dCPi3AoNvtFR52A_sd2D-ezjJVvM";
    String sensor = "sensor=false";
    String parameters = str_origin + "&" + str_dest + "&" + sensor + "&" + str_mode + "&" + str_transit_mode + "&" + str_key;
    String output = "json";

    String url = "https://maps.googleapis.com/maps/api/directions/" + output + "?" + parameters;

    return url;
}
```

Figura 2.19: Método getURL

A pesquisa representa a linha de ônibus que o usuário deseja monitorar. Após verificar no banco de dados, são retornadas a localização atual do ônibus e o destino final do trajeto. Ambas são enviadas como parâmetros na URL para a requisição do trajeto.

```
String url = getUrl(origin, destination);
Log.d("onMapClick", url.toString());
FetchUrl FetchUrl = new FetchUrl();

FetchUrl.execute(url);
```

Figura 2.20: Objeto FetchUrl criado

Após a formação da URL, um objeto da classe *FetchURL* é criado e executado, passando como parâmetro a URL criada. Essa classe é responsável por chamar o método *DownloadUrl*, que é responsável por fazer de fato a requisição HTTPS, como mostrado na Figura 2.21.

```
private String downloadUrl(String strUrl) throws IOException {
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
    try {
        URL url = new URL(strUrl);
        urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.connect();
        iStream = urlConnection.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(iStream));
        StringBuffer sb = new StringBuffer();

        String line = "";
        while ((line = br.readLine()) != null) {
            sb.append(line);
        }

        data = sb.toString();
        Log.d("downloadUrl", data.toString());
        br.close();

    } catch (Exception e) {
        Log.d("Exception", e.toString());
    } finally {
        iStream.close();
        urlConnection.disconnect();
    }
    return data;
}
```

Figura 2.21: Método DownloadUrl

Para essas requisições são retornados objetos JSON de resposta, e estes devem ser analisados para que as informações desejadas sejam extraídas. Essa função é realizada pela classe *DataParser*, sendo esta responsável pelas análise das respostas das requisições. A Figura 2.22 mostra um trecho do código onde são retiradas algumas informações sobre a latitude e longitude do percurso requisitado.

A classe *DataParser* analisa o JSON de resposta da *Directions* API, adiciona algumas informações a alguns arrays estáticos da classe *BusLocation*, e retorna um *array* principal como resposta. Um método importante dessa classe é o *decodePoly*, método responsável por decodificar os pontos contidos na rota, para que seja possível o desenho correto do trajeto no mapa. Este método é apresentado na Figura 2.23.

Após todos esses procedimentos, é mostrado no mapa todos os marcadores, a localização atual dos ônibus, as paradas e o destino final da linha. Todos eles são clicáveis, cada um com uma função diferente. Ao clicar no destino final, é selecionada a informação de tempo estimado de chegada, retornada pela API. Ao clicar em cada uma das paradas, uma *thread* similar à explicada anteriormente é iniciada, com a alteração do destino final para a localização da parada selecionada. É mostrado então o tempo de chegada estimado à parada.

```
for(int i=0;i<jRoutes.length();i++){
    jLegs = ( (JSONObject)jRoutes.get(i)).getJSONArray("legs");
    List path = new ArrayList<>();
    List path1 = new ArrayList<>();

    for(int j=0;j<jLegs.length();j++){
        jSteps = ( (JSONObject)jLegs.get(j)).getJSONArray("steps");

        legs = ((JSONObject) jSteps.get(j));
        jDuration = legs.getJSONObject("duration");

        for(int k=0;k<jSteps.length();k++){
            String polyline = "";
            polyline = (String)((JSONObject)((JSONObject)jSteps.get(k)).get("polyline")).get("points");
            List<LatLng> list = decodePoly(polyline);

            for(int l=0;l<list.size();l++){
                HashMap<String, String> hm = new HashMap<>();
                hm.put("lat", Double.toString((list.get(l)).latitude) );
                hm.put("lng", Double.toString((list.get(l)).longitude) );
                path.add(hm);
            }
        }
        routes.add(path);
```

Figura 2.22: Trecho da Classe DataParser

```
List<LatLng> poly = new ArrayList<>();
int index = 0, len = encoded.length();
int lat = 0, lng = 0;

while (index < len) {
    int b, shift = 0, result = 0;
    do {
        b = encoded.charAt(index++) - 63;
        result |= (b & 0x1f) << shift;
        shift += 5;
    } while (b >= 0x20);
    int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
    lat += dlat;

    shift = 0;
    result = 0;
    do {
        b = encoded.charAt(index++) - 63;
        result |= (b & 0x1f) << shift;
        shift += 5;
    } while (b >= 0x20);
    int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
    lng += dlng;
    LatLng p = new LatLng((((double) lat / 1E5)),
            (((double) lng / 1E5)));
    poly.add(p);
```

Figura 2.23: Método decodePoly() da Classe DataParser

Ao se clicar em um marcador de ônibus, a classe *BusInfo* se inicia, com informações específicas deste ônibus. Assim como as outras classes, esta se inicia com o método *onCreate*, mostrado na figura 2.24, que aciona o banco de dados através do método *RetrieveBD*.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_bus_info);

    editTextAlerta = (EditText)findViewById(editText9);

    getSupportActionBar().hide();
    RetrieveBD();


}
```

Figura 2.24: Método onCreate da classe BusInfo

Este método faz uma conexão com o banco de dados, utilizando novamente o Parse, para indicar as informações presentes no mesmo. A Figura 2.25 mostra o código onde essas informações são retiradas do banco de dados, e apresentadas para o usuário através de campos *TextView* criados no arquivo XML.

```
placa = ("Placa: " + Onibus.getString("Placa"));
linha = ("Linha: " + Onibus.getString("LineID"));
motorista = ("Motorista: " + Onibus.getString("Motorista"));
modelo = ("Modelo: " + Onibus.getString("Modelo"));
velocidade = (double) Onibus.getNumber("Velocidade");
velocidadeMax = (int) Onibus.getNumber("VelocidadeMax");
infracoes = (int) Onibus.getNumber("Infracoes");

velocidade1 = (int)velocidade;
 editTextPlaca = (TextView) findViewById(textView3);
 editTextPlaca.setText(placa.trim());

 editTextLinha = (TextView)findViewById(textView5);
 editTextLinha.setText(linha.trim());

 editTextMotorista = (TextView)findViewById(textView10);
 editTextMotorista.setText(motorista.trim());

 editTextModelo = (TextView)findViewById(textView7);
 editTextModelo.setText(modelo.trim());

 editTextVelocidade = (TextView)findViewById(textView12);
```

Figura 2.25: Informações BusInfo

A Figura 2.26 a seguir mostra o método utilizado para se verificar e alertar na tela caso o ônibus esteja trafegando acima da velocidade permitida da via. Para este cálculo, é verificado a via que o veículo está transitando no momento, e então é comparada esta informação com o banco de dados, afim de identificar qual a velocidade máxima permitida nesta via. Caso o veículo esteja transitando acima da velocidade permitida, é gerado um alerta para o usuário.

```
if (velocidade1 > velocidadeMax) {

    infracoes = infracoes + 1;
    Log.d("Infracoes: ", Integer.toString(infracoes));

    editTextAlerta.setVisibility(View.VISIBLE);
} else {
    editTextAlerta.setVisibility(View.INVISIBLE);
}
```

Figura 2.26: Código que verifica infração de velocidade

É utilizado no fim do método *RetrieveBD* a função *handler*, para que este método seja iniciado periodicamente, neste caso a cada 5 segundos, como mostrado na Figura 2.27. Este método é importante para que as informações apresentadas sejam o mais próximo do real. Este período foi escolhido com o intuito de equilibrar o sistema, de forma que ele não possua um consumo excessivo de dados móveis e de performance dos aparelhos, mas que apresente as informações em um período aceitável.

```
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            RetrieveBD();
        }
    }, 5000); //10 segundos de atualizacao


}
```

Figura 2.27: Código função Handler

Por último, foram criadas as seguintes três classes para o correto funcionamento do serviço de Chat:

- Message: Esta classe foi criada para realizar a comunicação com o banco de dados.
- ChatActivity: É a principal classe do serviço, onde são enviadas e recebidas as mensagens.
- ChatAdapter: Foi criada para organizar a estrutura do *Chat*, com as mensagens recebidas à esquerda e as mensagens enviadas à direita, além da imagem ao lado das mensagens.

As Figuras 2.28 e 2.29 mostram dois métodos presentes na classe *ChatActivity*, utilizados respectivamente para enviar e receber as mensagens.

```
public void enviarMsg(View view) {

    final String data = etMessage.getText().toString();

    if (data.length() == 0) {
        return;
    }

    Message message = new Message();
    message.setBody(data);
    message.setUserId(userId);

    message.saveInBackground(new SaveCallback() {
        @Override
        public void done(ParseException e) {
        }

    });

    etMessage.setText(null);
}
```

Figura 2.28: Código enviar mensagem

```
void refreshMessages() {

    ParseQuery<Message> query = ParseQuery.getQuery(Message.class);
    query.setLimit(MAX_CHAT_MESSAGES_TO_SHOW);
    query.orderByDescending("createdAt");

    query.findInBackground((messages, e) → {
            if (e == null) {
                mMessages.clear();
                mMessages.addAll(messages);
                mAdapter.notifyDataSetChanged();
                if (mFirstLoad) {
                    rvChat.scrollToPosition(0);
                    mFirstLoad = false;
                }
            } else {
                Log.e("message", "Error Loading Messages" + e);
            }
    });
}
```

Figura 2.29: Código receber mensagen

Capítulo 3

Funcionamento dos Aplicativos

3.1 Resultado e Análise

Este aplicativo foi criado com o intuito de disponibilizar informações sobre o veículo em tempo real, armazenando-as em um banco de dados. Ele funciona em um smartphone que deve ser instalado dentro do ônibus que, através da sua conexão com a internet, envia a localização precisa do veículo e sua velocidade, através da rede móvel e do GPS do celular. Essas informações são enviadas com um intervalo de 5 segundos, deixando assim o banco de dados atualizado com uma precisão considerável.

Sua tela principal apresenta dois campos, para serem informados uma única vez, as informações individuais do ônibus onde o *smartphone* esteja instalado. O primeiro campo identifica a linha do ônibus (cada linha pode possuir vários ônibus). O segundo campo identifica a ID do veículo (essa informação é única de cada ônibus). Ambos devem ser preenchidos assim como as informações do ônibus no banco de dados, para que o aplicativo de Gestão da Frota funcione corretamente. Os dados devem ser preenchidos por um responsável da empresa licitante, que após clicar no campo "INICIAR", são enviados de forma ininterrupta para a nuvem.

Após esse procedimento, é aberta uma nova tela com as informações que estão sendo enviadas ao banco de dados no momento. São exibidas a latitude, longitude, e velocidade estimada. É exibido também um botão de "PARAR", que possui a função de interromper o processo de captura dessas informações e de atualizar o banco de dados. O correto funcionamento desse aplicativo é fundamental para o trabalho de gestão dos ônibus. A parte inferior direita desta tela possui um botão, com um ícone de nuvem, onde é possível iniciar uma conversa com o gestor. Esta função será explicada com mais detalhes no Aplicativo Gestor. As Figuras 3.1 e 3.2 mostram a tela do aplicativo com as funcionalidades descritas.

Figura 3.1: Tela inicial Transmissor

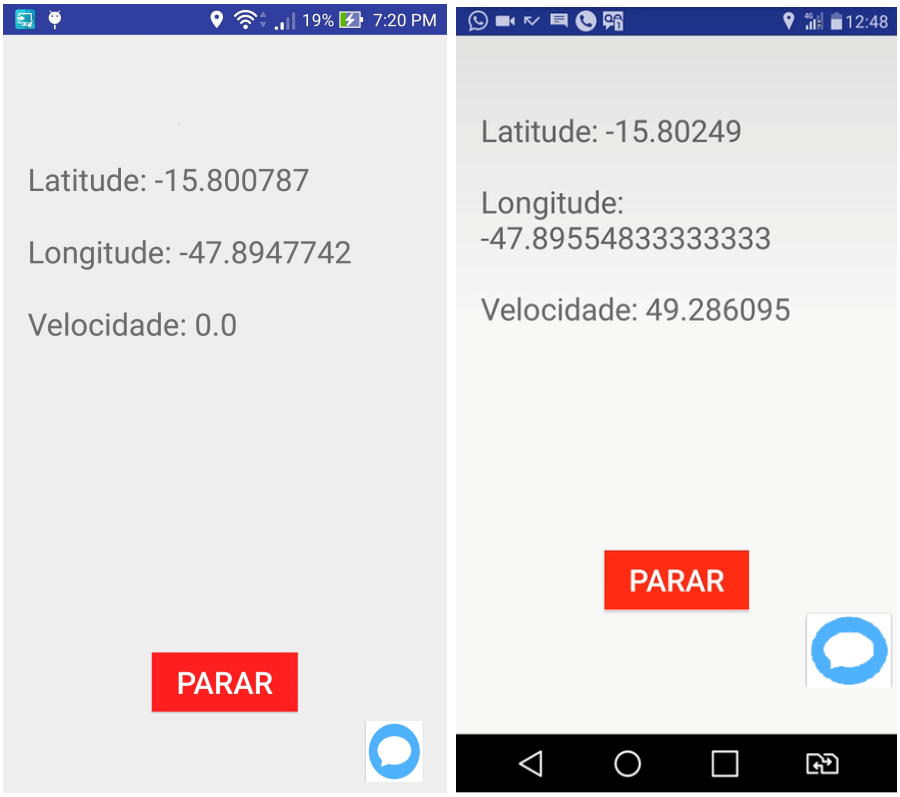

Figura 3.2: Informações atualizadas Transmissor

3.2 Aplicativo Gestor

Este aplicativo será utilizado por um gestor responsável por monitorar a frota de ônibus da empresa. Ele deve ser de fácil manuseio e rápido, para que seja possível realizar todas as suas funções de acompanhamento da frota em tempo real. Os aspectos de cor e layout foram levados em consideração também, para uma melhor experiência do usuário.

A página inicial do aplicativo é a tela de *Login*, onde são exigidas as informações de usuário e senha. Este passo é importante para que não seja garantido acesso de informações restritas da empresa e da frota a pessoas não autorizadas. Ao digitar suas informações pessoais, é verificado no banco de dados na nuvem a veracidade das mesmas, através do campo usuário, que deve ser igual ao digitado, e ao campo senha. Após digitar o usuário e senha, deve-se clicar no botão de cor azul "INICIAR" para se iniciar o aplicativo.

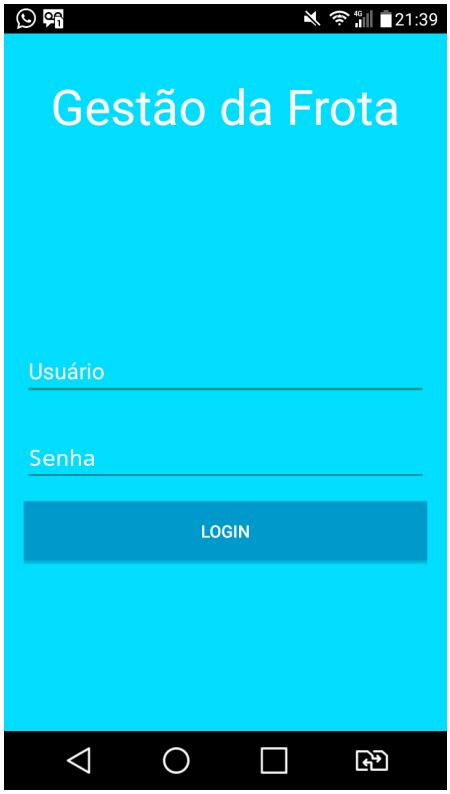

Figura 3.3: Tela Login App Gestão

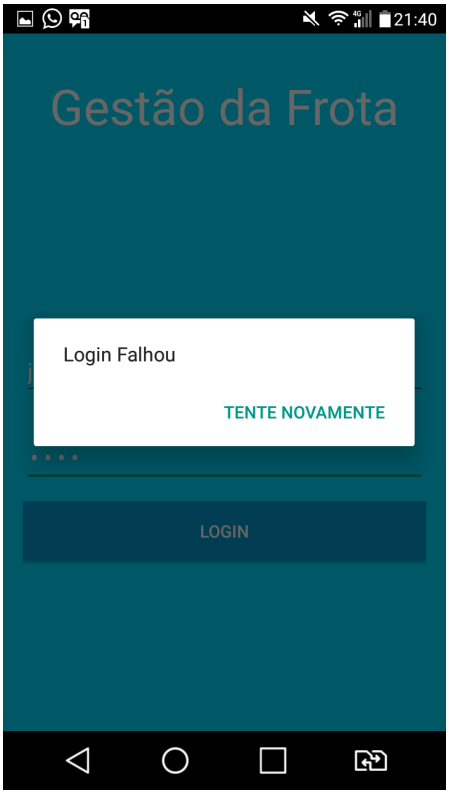

Figura 3.4: Erro de login App Gestão

Após o *login*, o usuário é redirecionado a um mapa, centralizado na região do Distrito Federal, mas sem um marcador com algum ponto de referência. É possível, através de alguns toques na tela, alterar a área visualizada, assim como o zoom na imagem. Esta tela apresenta no canto inferior direito um botão de *Logout*, que bloqueia a aplicação e retorna à tela de *Login*. Este procedimento deve ser realizado todas as vezes que o usuário terminar seu trabalho. No topo da tela encontra-se o campo de busca, onde deve ser digitado a linha de ônibus desejada.

Figura 3.5: Tela Mapa App Gestão

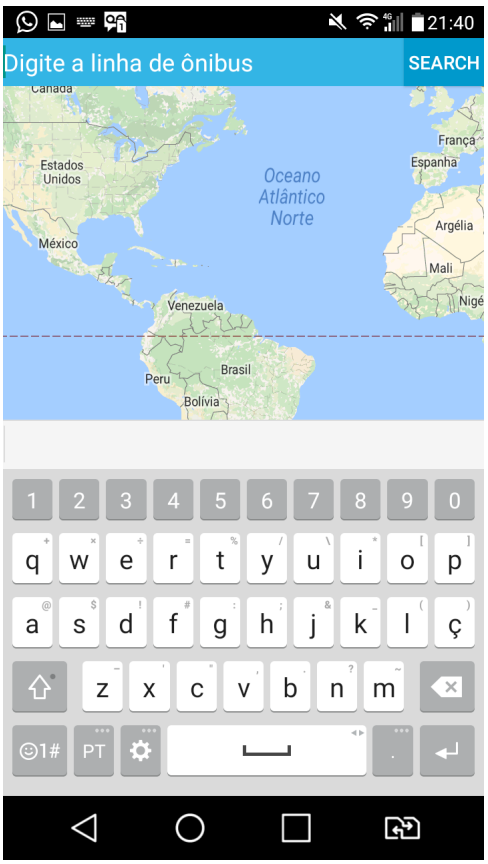

Figura 3.6: Pesquisa da linha de ônibus

Caso a linha digitada esteja presente no banco de dados, são retornadas ao aplicativo a Latitude e Longitude de todos os ônibus dessa linha, que estejam em funcionamento, e também a Latitude e Longitude do destino final da linha. Esta informação é fixa e deve ser inserida no banco de dados pelo gestor, enquanto a informação da localização de cada ônibus é atualizada em tempo real pelo aplicativo de Monitoramento. Quando retornadas essas informações, o aplicativo envia uma requisição HTTPS à Google Maps Direction API, solicitando as informações sobre a rota em questão. É retornado pela Google Maps Direction API, em formato JSON, diversas informações, sendo as principais: as rotas de diversas linhas de ônibus até o destino, e o tempo estimado de chegada para o destino. É feita uma análise no aplicativo e a rota escolhida é apresentada, específica da linha digitada previamente.

Estão cadastradas e são retornadas ao aplicativo também as informações referente à Latitude e Longitude de cada parada presente em toda a rota da linha de ônibus digitada. Essas localizações estão presentes no banco de dados, e não são retornadas pela Google Maps Directions API. Por isso, é necessário um trabalho prévio de inclusão da localização de cada parada de ônibus no banco de dados, ao se cadastrar uma nova linha de ônibus. Com essas informações, é mostrado no mapa a localização de cada parada de ônibus, com a identificação de um ícone específico. Ao se clicar em uma parada, é enviado uma requisição ao Google Maps Directions API, semelhante à explicada anteriormente referente ao destino final, com a diferença da localização final da rota desejada não ser a da linha, e sim da parada de ônibus que foi clicada. Com isso é retornado ao aplicativo o tempo estimado até essa parada específica. A Figura 3.6 mostra o tempo estimado sendo apresentado ao usuário.

Com essas informações de tempo de chegada estimado é possível fazer uma análise em tempo real do andamento dos ônibus da linha, como por exemplo quanto tempo cada ônibus está desviando do previsto de chegada ao destino final, e se existe algum trecho que requer mais atenção, devido a maiores atrasos apresentados. Está análise pode indicar um melhor dimensionamento e uma necessidade de mudança na logística dos veículos da frota, para melhorar a alocação dos recurso disponíveis e suprir as necessidades da população que utiliza este tipo de transporte.

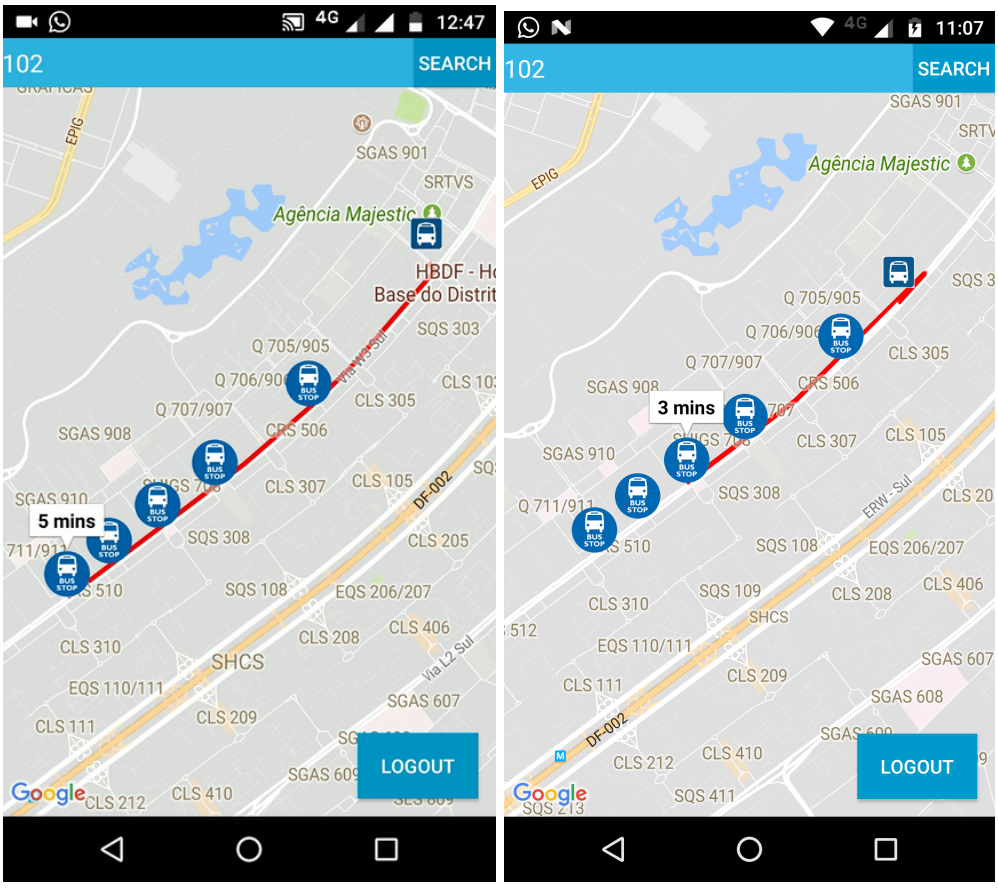

Figura 3.7: Tempo estimado até a parada de ônibus

Ao se clicar no ícone de algum dos ônibus mostrados no mapa, são abertas informações específicas sobre o mesmo. Cada informação está presente em uma coluna do banco de dados, e estas devem ser inseridas ao se cadastrar novos ônibus no sistema. Com elas é possível ter um controle mais próximo da frota. Nesta tela também é apresentada ao usuário a velocidade instantânea do veículo, informação esta enviada em tempo real pelo aplicativo de Monitoramento. A cada requisição ao banco de dados retornada com esta velocidade e com a localização atual do ônibus, é calculada no aplicativo a velocidade máxima permitida na via, através de informações previamente cadastradas no banco de dados. Caso a velocidade do veículo esteja acima da velocidade da via, é apresentado na tela um alerta, indicando que o motorista cometeu uma infração. A cada nova infração, é acrescido um ponto em uma coluna chamada "Infrações" no banco de dados. Com isto, é possível se ter um controle da quantidade de violações de velocidade cometida por cada motorista em cada ônibus.

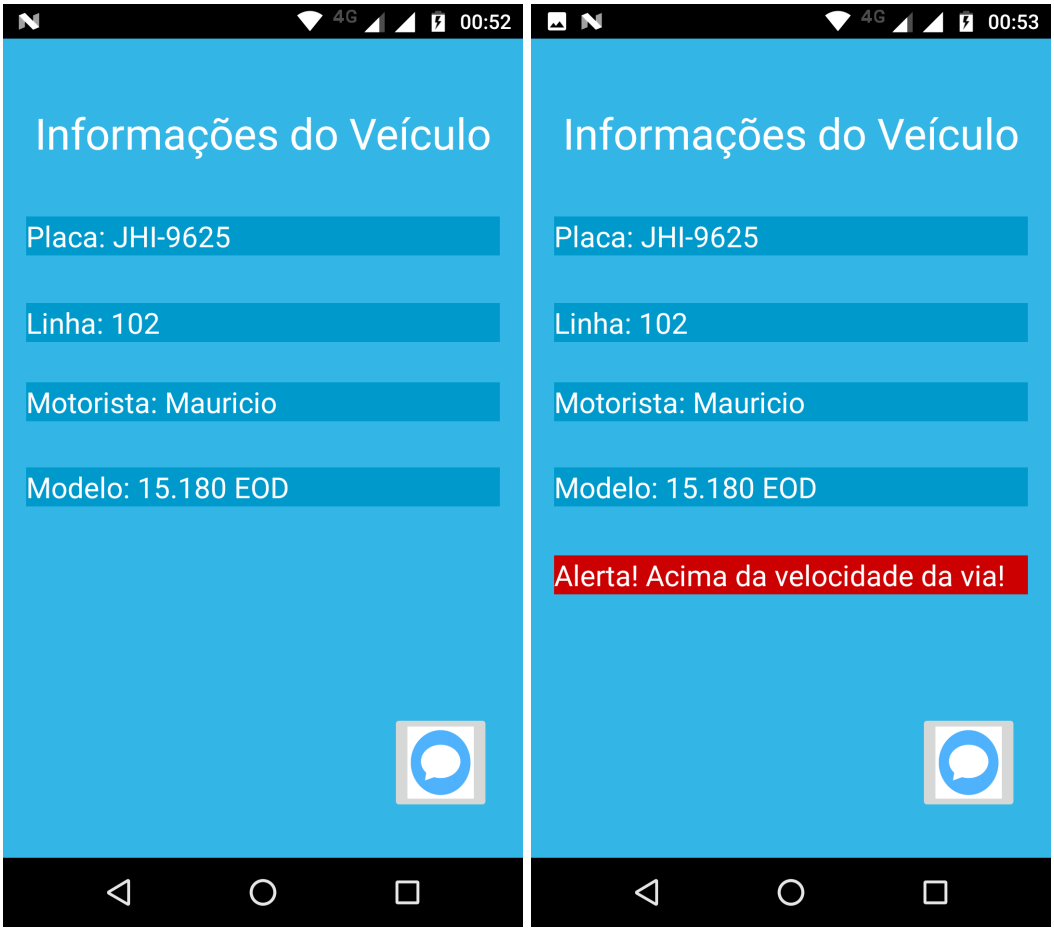

Figura 3.8: Tela de Informações App Gestão

Como última funcionalidade do aplicativo, ao se clicar no ícone de mensagem, como visto na Figura 3.8, é aberta uma tela de *Chat*, onde é possível iniciar uma conversa com um representante dentro do ônibus, como o motorista ou cobrador. Esta funcionalidade foi criada com o intuito de se estabelecer uma comunicação rápida entre uma pessoa no veículo e o gestor, a fim de resolver pequenos problemas que porventura possam ocorrer.

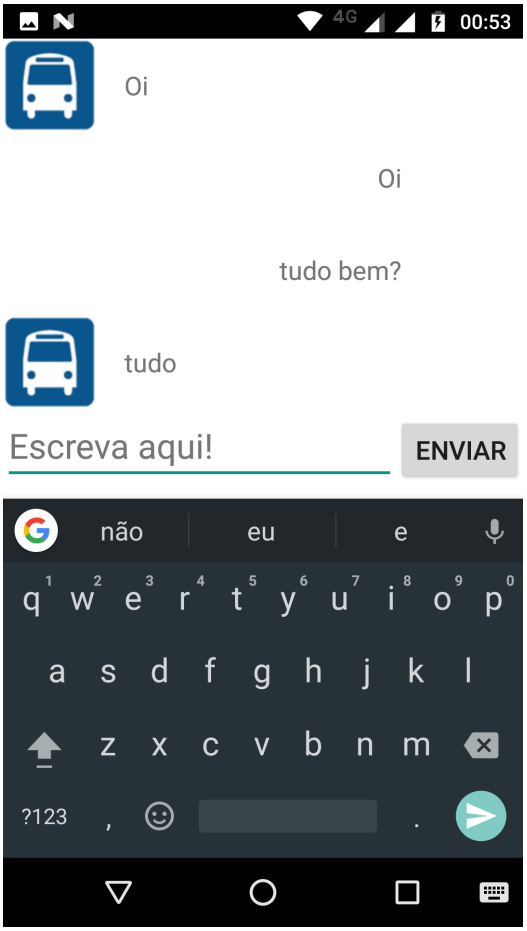

Figura 3.9: Tela Chat

Capítulo 4

Conclusão e Trabalhos Futuros

O sistema de transporte público no Brasil é considerado um dos serviços públicos com mais problemas e que gera constantes reclamações. Uma solução personalizada para este tipo de serviço pode gerar resultados positivos para as empresas responsáveis, para o governo e principalmente para a população. Com este aplicativo seria possível analisar a frota em tempo real, de forma a identificar mais facilmente possíveis falhas no planejamento do sistema, otimizando a alocação dos veículos e dos recursos. Além disso, com o controle de infrações, juntamente com a organização das informações de cada veículo, pode-se gerar uma redução em alguns custos desnecessários da empresa, impactando assim no serviço oferecido à população.

A seguir são listados alguns objetivos de trabalhos futuros:

1. Otimização do aplicativo: Devido à constante troca de informações entre o banco de dados e os dois aplicativos, este sistema pode ser otimizado, com um desenvolvimento visando um menor consumo de bateria e de dados móveis. Por utilizar algumas funções periodicamente, e por ser utilizado de forma ininterrupta, o consumo dos recursos será muito alto. Com um desenvolvimento mais eficiente e voltado para a performance, é possível otimizar estes recursos e este sistema no geral.

2. Inclusão de outras funcionalidades: Estes aplicativos podem ser melhorados com a adição de diversas funcionalidades interessantes, que podem agregar o sistema criado, tais como: indicador de manutenção preventiva e alerta de veículo transitando fora do trajeto previsto. Além destas, a inclusão de sensores conectados aos ônibus e ao aplicativo, como um sensor de segurança, que seria acionado em caso de assalto, ou um sensor conectado no ônibus, para verificar as informações do odômetro do veículo, podem ser úteis em projetos futuros de aprimoramento do sistema.

3. Análise dos dados com BI: Esta é uma tecnologia para análise e processamento de dados para se gerar informações de ações que podem ajudar corporações, desenvolvedores e gestores a tomarem decisões que ajudam o negócio a se desenvolver. A implantação deste projeto iria gerar uma grande quantidade de dados acerca das rotas dos ônibus no Distrito Federal, como

o tempo de duração de uma linha analisada em diferentes horários do dia, e informações de tráfego. A utilização de BI, na análise desses dados, pode ajudar a identificar horários onde a quantidade de ônibus de uma linha está sendo insuficiente, ou trechos que estão gerando atrasos e que podem ser substituídos por outras rotas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] DEVELOPERS.GOOGLE. *A Google Maps Directions API.* Disponível em: https://developers.google.com/maps/documentation/directions/intro?hl=pt-br. Acessado em 18/06/2017.

[2] DEVELOPERS.GOOGLE. *Google Places API: Getting Started.* Disponível em: https://developers.google.com/places/android-api/start. Acessado em 18/06/2017.

[3] DEVCENTER.HEROKU. *Heroku Command Line.* Disponível em: https://devcenter.heroku.com/articles/heroku-command-linedownload-and-install. Acessado em 08/05/2017.

[4] MAROTTO, Fosco. *What is Parse Server?.* Disponível em: http://blog.parse.com/announcements/what-is-parse-server/. Acessado em 28/04/2017.

[5] PARSE. *Migrating an Existing Parse App.* Disponível em: https://parse.com/migration. Acessado em 23/05/2017.

[6] OI. *Oi Gestão de Frotas.* Disponível em: http://oigdf.pvinova.com.br/login. Acessado em 15/06/2017.

[7] VIVO. *Vivo Gestão de Frotas.* Disponível em: http://www.vivo.com.br/portalweb/appmanager/env/web?_n
Acessado em 15/06/2017.

[8] IVECO. *Iveco Frota+.* Disponível em: http://ivecofrotafacil.com.br/demo.html. Acessado em 15/06/2017.

[9] DIWAKAR, Sapan. *Recycler View Item Click Listener.* Disponível em: http://sapandiwakar.in/recycler-view-item-click-handler/. Acessado em 08/06/2017.

[10] LACKER, Kevin. *Moving On.* Disponível em: http://blog.parse.com/announcements/moving-on/. Acessado em 04/05/2017.

[11] HU, Roger. *Building Simple Chat Client with Parse.* Disponível em: https://github.com/codepath/android_guides/wiki/Building-Simple-Chat-Client-with-Parse. Acessado em 07/06/2017.

[12] WHITING, Timothy. *Setting up your own Parse Server.* Disponível em: https://medium.com/@timothy_whiting/setting-up-your-own-parse-server-568ee921333a.jw0tuuo88. Acessado em 14/05/2017.

[13] VOGELLA. *Android Development.* Disponível em: http://www.vogella.com/tutorials/android.html. Acessado em 23/06/2017.

[14] MATHEW, George. *Drawing driving route directions between two locations using Google Directions in Google Map Android API V2.* Disponível em: http://wptrafficanalyzer.in/blog/drawing-driving-route-directions-between-two-locations-using-google-directions-in-google-map-android-api-v2/. Acessado em 07/05/2017.

[15] REGMI, Pacific. *Google Maps Android API Adding Search Bar: Part 3.* Disponível em: http://www.viralandroid.com/2016/04/google-maps-android-api-adding-search-bar-part-3.html. Acessado em 15/05/2017.

[16] PARSEPLATFORM. *Android Guide.* Disponível em: http://parseplatform.github.io/docs/android/guide/u objects. Acessado em 14/06/2017.